Detecting Ambiguities: An Optimistic Approach to Robustness Problems in Computational Geometry‡

Beat Bruderlin
Computer Science Department
University of Utah
Salt Lake City, UT 84112,
bruderlin@cs.utah.edu
Technical Report Number: UUCS-90-003

April, 1990

‡This work was supported in part by NSF grant # DDM-8910229

Detecting Ambiguities: An Optimistic Approach to Robustness Problems in Computational Geometry‡

Beat Bruderlin
Computer Science Department
University of Utah
Salt Lake City, UT 84112,
bruderlin@cs.utah.edu

Abstract

Computational geometry algorithms deal with geometric objects, usually represented by coordinates in an n-dimensional Euclidean space. Most efficient algorithms implement geometric operations as floating point arithmetic operations on the coordinates. Since floating point numbers can only approximate the "real" world, these operations often lead to topologically inconsistent results, especially when degenerate cases are handled. Recently, a variety of methods have been developed to cope with this, so called, robustness problem. This paper describes a new approach based on the optimistic assumption that in the majority of cases the decisions can be made consistently, even with imprecise data. Degenerate cases are decided with some tolerance. A test is applied for detecting when decisions made by the algorithm that logically depend on each other are inconsistent due to ambiguities arising from the approximation. In case of ambiguities, the inconsistencies can be resolved by increasing the tolerance. The proposed ambiguity test can be carried out in constant time whenever a decision is made during computation. Therefore, this method does not change the asymptotic complexity of the underlying algorithm in most practical cases, which is a clear advantage over previous approaches.

1 Introduction

Efficient algorithm design is one of the major topics in computational geometry. The nature of computational geometry algorithms is that they consist of operations on geometric objects (e.g., intersecting lines and planes, positioning objects, etc.) and of decisions based on geometric relations of such objects (e.g. that two lines are parallel or not, points are coincident with lines, etc.). The analysis of the complexity of most algorithms relies on the fact that these basic operations and comparisons can be computed in constant time. However, this is only the case if the underlying geometric primitives are represented with

‡This work was supported in part by NSF grant # DDM-8910229

constant precision. Many algorithms in computational geometry therefore use a floating point representation. Floating point numbers are very flexible in representing large domains of real values. Floating point arithmetic operations are carried out in constant time, and they are efficiently implemented in the hardware of most computers. On the other hand, floating point numbers only approximate the domain of real numbers. This causes severe difficulties with the interpretation of the data, when branching decisions in an algorithm are based on the comparison of approximate data. The result of the algorithm can be inconsistent.

Robustness against inconsistent interpretation is an important requirement for most algorithms involving numerical data. In computational geometry, degenerate cases, such as parallel or collinear lines, coincident points, etc. already make the structure of algorithms more complicated. Together with the inaccuracies due to the limited precision of floating point variables and operations degenerate cases can lead to unpredictable and inconsistent results. Some recent publications address the robustness problem in computational geometry [HHK 88], [HOF 89], [SES 88], [YAP 88], [GSS 89], [OTU 87]. Most of the proposed solutions drastically increase the complexity of the algorithm, or they restrict the domain of representable objects. It seems necessary to trade in speed and expressiveness for robustness.

We present a new, more pragmatic approach to the robustness problem in computational geometry which does not have most of the drawbacks of previous approaches. The method applies simple reasoning on the uncertainty domain of the data, and thus finds out whether or not we can trust the decisions made during computations. It detects ambiguities that may have lead to decisions that are inconsistent with decisions made elsewhere in the algorithm. No explicit reasoning about the dependency of a decision from other decisions is required. The method can be applied to most existing computational geometry algorithms without changing their logical structure, and without affecting their asymptotic time complexity in the majority of cases. Moreover, it imposes almost no restriction on the intended data model, in the sense that it handles both, degenerate and generic cases. The underlying operations can all be implemented in limited precision floating point arithmetic.

2 Handling Approximate Data

Representing real values, such as point coordinates, angles, etc. by floating point variables poses some inherent problems:

a) Floating point variables are only an approximation of the Euclidean space $\Re^n$, and b) arithmetic operations cause round-off errors that accumulate with the number of successive

operations carried out on these numbers. As a consequence, we can have only limited faith in data represented by floating point values. Each value is surrounded by a region of uncertainty ε which is a function of the accuracy of the data representation and the preceding operations carried out on the data.

> **Example:** Point sets in $\Re^2$
> For a given, finite set S of points we try to find pairs of coincident points (for each point we have the computed x/y-coordinates and a radius of error ε). We can observe the following: If for two points the ε regions don't overlap we can assume that the points do not coincide in reality. On the other hand, if the ε regions overlap there are two possible interpretations: a) the points are meant to be identical or they were meant to be distinct but happen to be less than ε apart. The uncertainty in the representation makes it impossible to prefer one interpretation from the other; it remains ambiguous.

In computational geometry algorithms such decisions as whether two points are coincident, or not, whether two lines are collinear, parallel (degenerate cases), or whether they intersect (generic case), etc., are very important. Making "wrong" decisions usually has an unpredictable outcome. Therefore, we must find a way of handling decisions on approximate data consistently. A heuristic that is applied by many algorithms using floating point data representation is the *tolerance paradigm.*

2.1 The Tolerance Paradigm

We define a distance d (in Euclidean metric) between two geometric objects r_1 and r_2:

$$d = |r_1 - r_2|$$

If the distance d is less than the uncertainty of the data ε we arbitrarily decide that $d = 0$ (i.e. the two objects coincide), otherwise we say that they are distinct. For each geometric object we have to estimate an upper bound for ε from the operations that were applied to compute its coordinates. There exists a variety of techniques for estimating errors, such as interval arithmetic, condition numbers, etc. We do not further discuss these techniques here, but instead refer to the literature (see, e.g. [ALH 83]).
The tolerance paradigm is a compromise applied by many algorithms; it works fine in most cases. However, it also has some severe consequences, as the following examples show:

2.2 Consequences of the Tolerance Paradigm (Examples):

a) In an assembly consisting of small and big objects: When the dimensions of a small object is smaller than the ε uncertainty of the vertices of some big object then all

vertices of the small object coincide in one point (compared to the big one). Therefore, "features" that are smaller than the data uncertainty disappear in this interpretation.

b) When we want to find out which of 3 points (see fig. 1) coincide according to the tolerance paradigm, we might find that the ε regions of points A and B overlap. Therefore, we would determine A coincides with B. For the same reason we would say that B coincides with C. The ε regions of A and C don't overlap. We therefore conclude that A and C do not coincide.

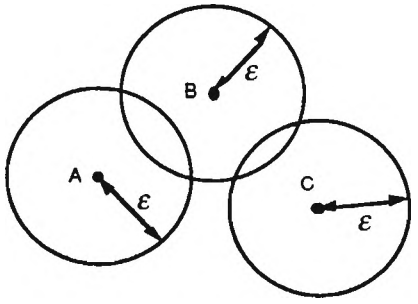


Fig. 1. Comparing three points with uncertainty.

On the other hand, coincidence is an equivalence relation. From the transitivity property of equivalence relations we must conclude that, if A coincides with B, and B coincides with C then also A coincides with C, which contradicts the previous finding. We conclude that by applying the tolerance paradigm for deciding coincidence we are sometimes inconsistent with the intended model (the tolerance paradigm does not preserve the transitivity property of the incidence relation).

c) When we approximate a circle by a polygon the points on the circle will always have a certain distance δ from the polygon. We try to make δ smaller than the ε uncertainty of the circle. This can be done by adding more and more points to the polygon. By doing so, also the angles between adjacent line segments in the polygon become smaller. At some point the angles become smaller than the ε resolution for angles. Therefore, we assume that adjacent line segments are collinear. However, since collinearity is transitive, all the lines in the polygon will be collinear. This, in turn, means that all lines in the plane are parallel, and all circles are straight lines. This, again, contradicts the intended model.

The conclusion we draw from the above examples is that certain objects cannot be consistently represented with limited precision arithmetic, even together with the tolerance paradigm.

Note! These problems can occur, independent of the size of ε, and therefore, independent of the precision that is used for the computations, as long as the precision is finite. Therefore, estimating the range of uncertainty for the data is a necessary, but not sufficient requirement for consistently computing with floating point numbers.

There have been done some very different approaches to the robustness problem in computational geometry:

- Arbitrary precision rational arithmetic [SES 89]
- Reasoning [HHK 88], (see also [BCK 88], [BRU 88], [CHO 88])
- Perturbation of data to avoiding degeneracies [EDM88]
- Line cracking [MIL 88]
- ε-geometry [GSS 89]

An overview on a variety of methods can be found in [HOF 89]. A short discussion of some of them, in comparison with the approach presented here, is given in the concluding chapter of this paper.

3 Avoiding Inconsistencies by Detecting Ambiguities

The major problem arising from the approximation of data is that we cannot make a distinction between values that are meant to be equal, but differ by a small amount due to some round-off error, and values that are meant to be different by a very small distance in the configuration space, but the distance is smaller than the inaccuracy. We must accept that the two cases are indistinguishable, and therefore must be handled identically, for instance, by deliberately interpreting small distances as zero (as this is postulated by the tolerance paradigm). However, in some border cases, when the difference is just about ε, the outcome of the decision becomes unpredictable. At some instance we decide a very small value to be zero, at some other instance the value is just big enough to escape this criterion; it is left unequal zero. When the two cases are logically connected (like in the case of transitivity of point coincidence) the result is inconsistent and the outcome of the algorithm is unpredictable.

In the following, we present a new approach that is based on the fact that decisions cannot be made independently of each other, in the sense that previous decisions never should contradict later decisions when they logically depend on the previous decision. Instead of

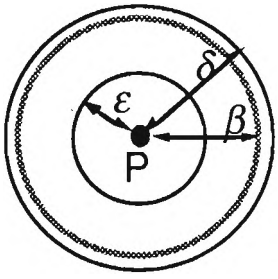
applying symbolic reasoning on the possibly involved logical dependencies of geometric relations (this would require the application of a usually complicated theorem proving program, and is often computationally expensive or even intractable) we reason about the influence each decision has on the uncertainty of the data. At each instance we make a three-valued decision (generic/degenerate/ambiguous). Later decisions may find that a previous 'degenerate' decision now results in 'ambiguous' but never indirectly implies 'generic'. A 'generic' decision may become 'ambiguous' later, but never implies 'degenerate'. By this we can guarantee that the outcome of the algorithm is never inconsistent. Either the algorithm runs successfully, i.e. no ambiguities are discovered, or we detect ambiguities, in which case we may try to rerun the algorithm with a different estimate for the ε values, and hopefully resolve the inconsistencies.

In the following we describe how geometric relations are computed under the assumption that the data has some uncertainty. We first describe the method at the example of point coincidence in 3-D Euclidean space. Later, we show how the method is generalized for other geometric objects, relations and dimensions.

4 Computing Point Coincidence Consistently

First we have to describe the data structure for points used by the approach. Each point object P is represented by three regions, ε, β, δ, as depicted in fig. 2. The interpretation of those regions is as follows: ε is the uncertainty of the data point P. This means that the point which is approximated by P is in reality somewhere within the sphere with center P and radius ε.

Fig. 2. Point with data with error
Tolerance region: ε
Buffer region: β
Distinction region: δ



When we compare P with a second point Q, and the two ε spheres of P and Q have a non empty intersection the two points could have the same location in reality, but they could also be distinct. We follow the tolerance paradigm, and determine that P and Q are equal. To consistently decide when two points are distinct we define a second region around each point called $\delta \supseteq \varepsilon$. We say that a point Q is distinct from a point P when its δ sphere has

an empty intersection with the δ sphere of P. To make sure that subsequent decisions are consistent with previous ones we need to keep some information about each decision made. In the following we describe the strategy of comparing points consistently:

Initially the ε region is a sphere with center P and a radius representing an upper bound for round-off errors in computing the position of the point P. δ must include ε (e.g. we can initialize the δ region to be a sphere with center P and twice the radius of the ε sphere). We also define a so called buffer region β that is between ε and δ, and separates the two (we initialize $\beta = \delta$). The role of the β region will be explained later. The following invariants must hold during the subsequent comparisons:

$$\varepsilon \subseteq \beta, \text{ and } \beta \subseteq \delta .$$

When comparing two points we update the ε, β, and δ regions in order to make sure that subsequent decisions do not contradict previous ones. When the ε regions of the two points overlap we arbitrarily decide that they are equal. The consequence of this decision is that each point can now be located everywhere in the uncertainty region of the other point. Therefore the new ε region of the two points will be the union of the two spheres [1]. In other words, not only geometric operations contribute to the uncertainty of the data, but also deciding geometric relations.

We consider two points to be clearly distinct if the δ region of one point does not overlap with the δ region of the other point. When we decide that two points P and Q are equal using the above criterion for equality we have to make sure that every point R that is equal to, say Q, is also identical to P, but also that every point Z that is distinct from, say P, is also distinct from Q, and vice versa, otherwise, we violate the transitivity of the equality relation. As we have seen before this behavior is not guaranteed by the tolerance paradigm. To explain the various cases that may occur we look at fig. 3. Points P and Q are shown with their ε and δ regions. P and Q are equal, because their ε regions overlap. When we compare P and Q separately with the points U, V and Z, we find that the δ region of Z has no overlap with the δ region of neither P nor Q.

---

[1] To make this operation geometrically simple we say the new uncertainty region ε of both points is the smallest sphere including the union of the two ε spheres.

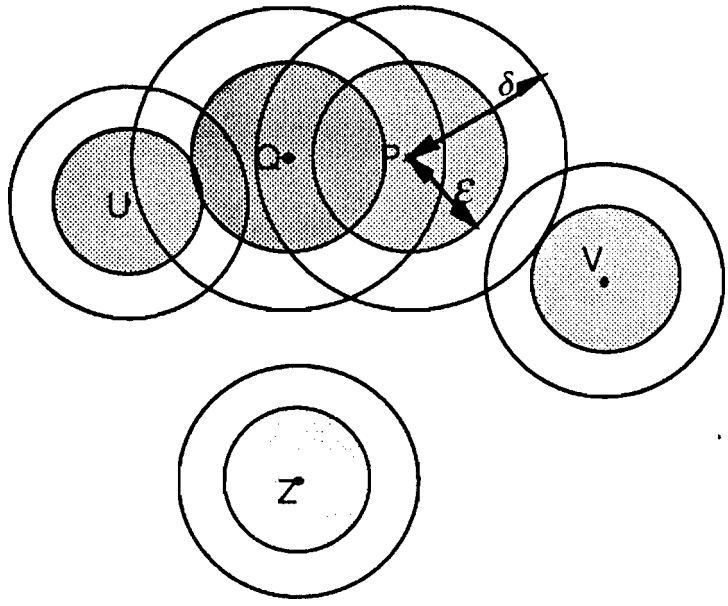


Fig. 3. Comparing several points with ε- and δ regions with each other.

It is therefore clearly distinct from both points, P and Q, which would be consistent with a later decision that P = Q. Point U is distinct from P but not from Q since its δ region overlaps with the δ region of Q. Therefore, a later decision that P = Q would contradict this decision. Similarly we would get a contradiction for point V which is distinct from Q but not from P. Therefore, a point is clearly distinct from two points P and Q (where P = Q), when its δ region does not overlap the with union of the two δ regions. Whenever we merge two points that we found to be equal, the new common δ region will be the union of the two previous δ regions [2]. We also must make sure that the ε region of some point R remains completely within the δ region of any point F_i that was found to be coincident with R. Otherwise, if ε of R grows beyond the boundaries of the δ region of F_i, it may happen, that some point S that was said to be distinct from F_i can be equal R which is a contradiction. This leads us to the role of the β region. Whenever we consider two points to be coincident their common β region will become the intersection of the two previous β regions [3] (which were originally set to δ). Thus, the β region of each point is contained in the intersection of the δ regions of all points that were compared and found to be coincident with it. When ε grows into β this means that the point in reality may exist outside the δ

---

[2] For simplicity we compute the new δ region to be the smallest sphere including the union of the two δ spheres.

[3] For simplicity we compute the new β region to be the biggest sphere included in the intersection of the two β spheres.

region of at least one of the other points that were considered to be coincident. In this case we report an ambiguity.
Note that after updating the ε, δ, and β regions, the spheres a not necessarily concentric any longer!

To formally prove that the method described here defines point coincidence consistently with the model of points in $\Re^n$ we first have to give some formal definitions.

Definitions (coincident, distinct, equivalence class, consistent set of points).
Two points P_j, P_k, are *coincident*: $P_j = P_k$, iff $\varepsilon_j \subseteq \delta_k \wedge \varepsilon_k \subseteq \delta_j$ [4)]
Two points P_j, P_k, are *distinct*: $P_j \neq P_k$, iff $\delta_j \cap \delta_k = \varnothing$

The points P_1, .. P_k, build an *equivalence class* $\underset{i=1}{\overset{k}{=}} P_i$, iff $\bigcup_i \varepsilon_i \subseteq \bigcap_i \delta_i$

A set of point S is *consistent*, if $\forall P_j, P_k \in S$, either $P_j = P_k$, or $P_j \neq P_k$.
Note, that this definition of consistent corresponds to the previously given definition for the data structure of points with the starting condition, $\forall P_i\ \varepsilon_i \subseteq \delta_i$, and the update operations upon pairwise comparison of points, if we don't find an ambiguous situation, i.e. as long

as $\varepsilon_i \subseteq \beta_i$, where $\beta_i = \bigcap_{P_j = P_i} \delta_j$.

Lemma 1 If a set of points S is consistent then coincidence of points is an equivalence relation for this set.
Proof of Lemma 1 We can easily show that from the above definitions follows:

$\forall P_j, P_k \in \left(\underset{i}{=} P_i\right) \subseteq S$: $P_j = P_k$, and $\neg\exists\ P_j, P_k \in \left(\underset{i}{=} P_i\right)$: $P_j \neq P_k$. Therefore, $P_j \neq P_k$ is equivalent to $\neg\ (P_j = P_k)$. The transitivity, reflexivity, and symmetry properties of the coincidence relation can be directly derived. □

---

4) Note that the condition for coincidence is weaker than the one given previously. It is no longer necessary that the ε regions overlap.

5 Computing Relations of Higher Dimensional Objects Consistently

The method described for the coincidence relation of points can be generalized to relations of other geometric objects (e.g parallelism of lines and planes, collinearity, coplanarity, incidence of points with lines or planes, or lines with planes, etc.). We describe the method in more detail for the incidence relation of points and lines, and the collinearity of lines in 2D. We then mention how this method is extended to three-dimensional linear objects (points, lines, and planes), or even higher dimensions.

5.1 Two-dimensional Objects and Relations

Between points and lines we can determine the so called incidence relation. When we have incidence of one point with two lines, either the two lines are identical or the point is an intersection point of the two lines. Fig. 4 shows three pairwise intersecting lines. Each line is shown with an ε environment.

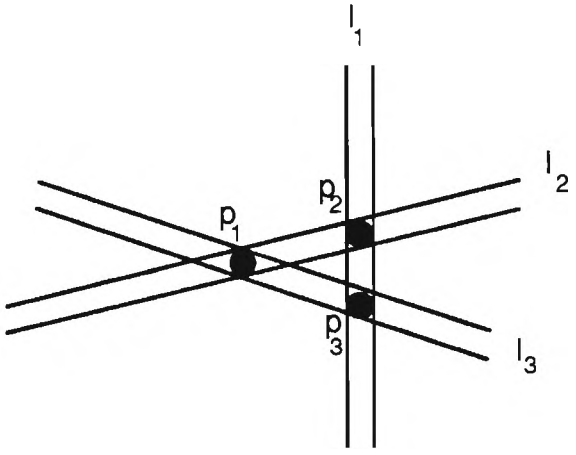


Fig. 4. Intersecting 3 lines with uncertainty.

We can observe the following possibilities. Either p_1, p_2, and p_3 are pairwise distinct, or if $p_2 = p_3$, then either $p_2 = p_3 = p_1$, or $l_2 = l_3$. Otherwise the relations are inconsistent. Again we see that the relations depend on each other. Dependencies of incidence relations between points and lines can become arbitrarily complex. A little more complicated example is shown in fig. 5.

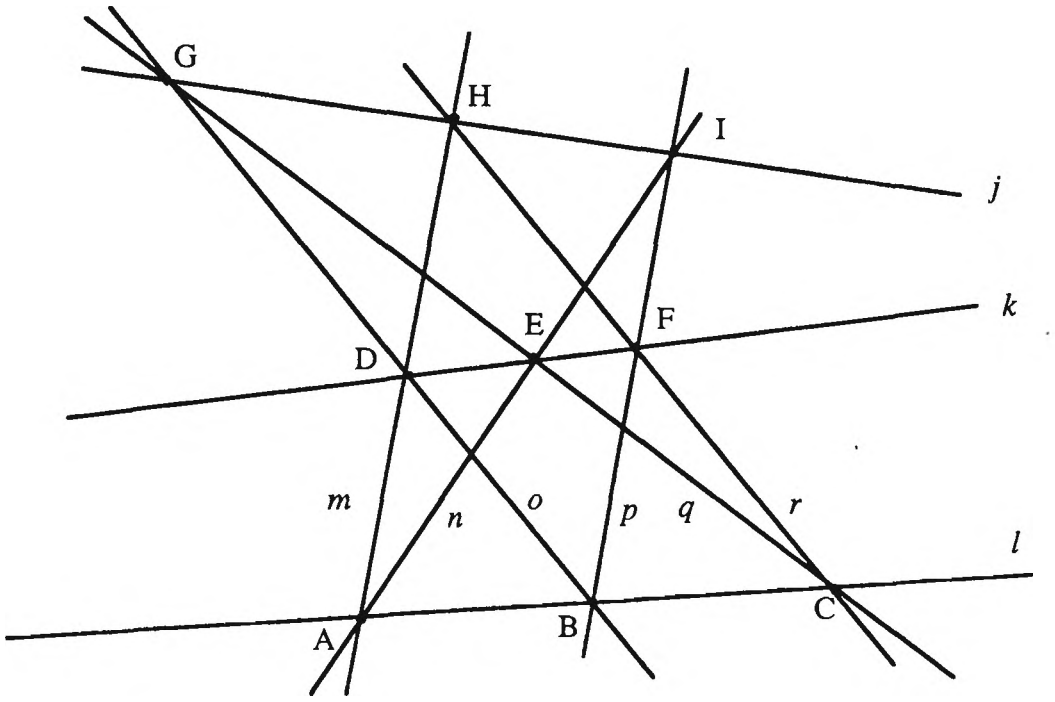


Fig. 5 Nine intersecting lines (Pascal's theorem).

The figure consists of 9 lines *j*, *k*, *l*, *m*, *n*, *o*, *p*, *q* and *r* . At each of the eight points A, B, C, D, E, F, H and I, three of the lines intersect. According to Pascal's theorem (in the version given here) also lines o, q and j must intersect in one point (point G in drawing).

For a given set of 9 lines we want to verify that it is not in conflict with Pascal's theorem, i.e., either at all nine points A, B, C, D, E, F, H, I and G three of the lines intersect, or in at least two cases the three intersecting lines create three intersection points. In the latter case the hypothesis is not fulfilled, therefore, the theorem does not apply.

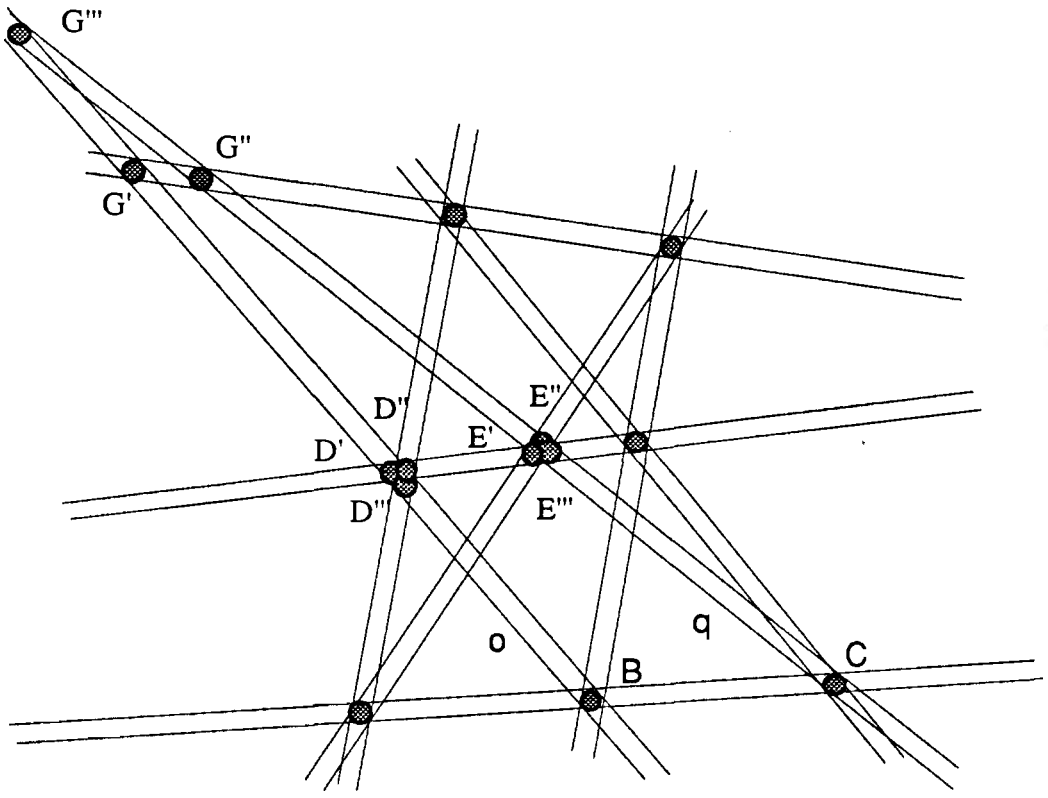


Fig. 6 Disproving Pascal's theorem with approximated points and lines.

When the lines are approximated by floating point values we can think of them as thick lines, as shown fig. 6. The lines in this figure correspond to the lines in fig. 5, but we slightly rotated the two lines *o* and *q* around the points B and C, respectively. Therefore, the previous points D, E, and G aren't single points any longer. For each of them we have three points instead. Since the lines are approximated, also the intersection points can only be computed with limited accuracy (they are approximated by circles). According to the tolerance paradigm the three points D', D'' and D''' (replacing point D) would be considered coincident. The same is the case with points E', E'' and E'''. In place of point G we now have three points G', G'' and G''' with disjoint ε regions. Since they are too far apart they will not be considered coincident according to the tolerance paradigm. In all but one case three lines intersect in one point. Therefore, this interpretation violates Pascal's theorem.

The conclusion from this example is that, determining relations of points and lines using the tolerance paradigm is not necessarily consistent with the theory of Euclidean geometry. This is no surprise since we already showed that the basic axioms for point coincidence may be violated. Also, we conclude from this example that, if we used symbolic reasoning

for obtaining a consistent interpretation of geometric relations we would need a powerful theorem prover capable of proving, for instance, Pascal's theorem. Already this relatively simple example is much too time consuming to do with automated theorem proving methods (see e.g. [BCK88], [BRU 88], [CHO 88]). The theorem proving part alone would take many times more computing cycles than the original algorithm which just intersects 9 lines with each other, and finds pairs of coincident points. In practical applications we usually deal with thousands of lines, and theorem proving becomes intractable. In general, it is not always easy to determine what has to be proved to make the underlying model of an algorithm consistent; each model has its own theory.

In the following we want to extend the idea of having ε, β, and δ regions to line objects, and make incidence and collinearity decisions consistent. Fig. 7 shows the ε environment for a line in two dimension. The line actually represents a line segment of finite length. The ε region of a line segment is a rectangle that has the same orientation as the line and width ε. The rectangle has to be long enough to contain all points incident with that line segment.

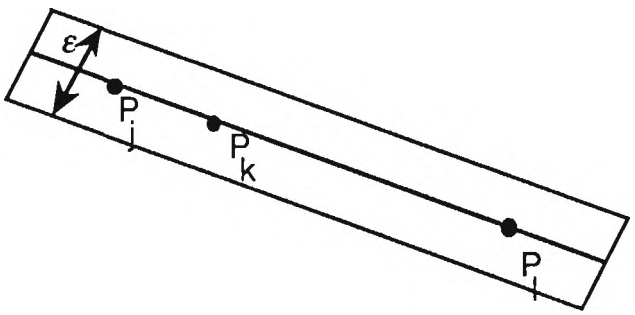


Fig. 7. The ε-environment of a line in 2D.

As it was the case for points we define β and δ environments also for line segments. The δ region is initially a rectangle of the same length and orientation as the ε rectangle, but with greater width. The β region is initially the same as the δ region. The procedure to determine collinearity of two lines is done similarly to the computing of the coincidence relation for two points. We define that two lines are *not* collinear if they either intersect, or if they are parallel and their δ regions have a non empty intersection. Otherwise, we assume they are collinear in which case the new ε region will be the union of the two individual ε regions (approximated by the smallest rectangle with the average orientation of the two lines containing both previous ε rectangles. See fig. 8a). An intersection operation is done with the β regions (approximated by a rectangle of the same orientation and length as the ε

rectangle, and just wide enough to exclude all the corner points of the two previous β rectangles. See Fig 8b), and a union operation is carried out with the δ regions. Again, the ε region is not allowed to grow into the β region, otherwise it is inconsistent. Two line segments are therefore defined to be collinear, if the ε and β regions can be updated consistently ($\varepsilon \subseteq \beta$) with the described procedure.Two parallel lines are separate, if their δ regions don't overlap.

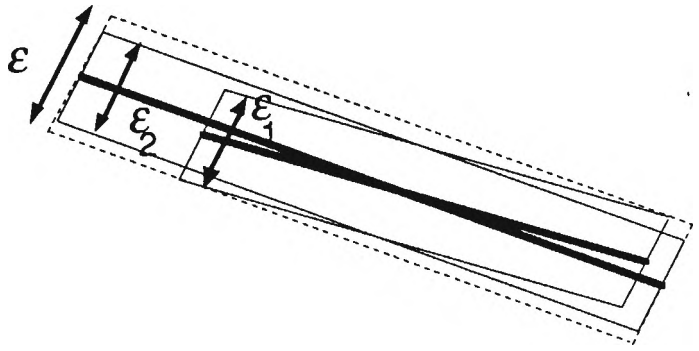


Fig. 8a. Updating the ε region, after merging 2 collinear lines in the plane.

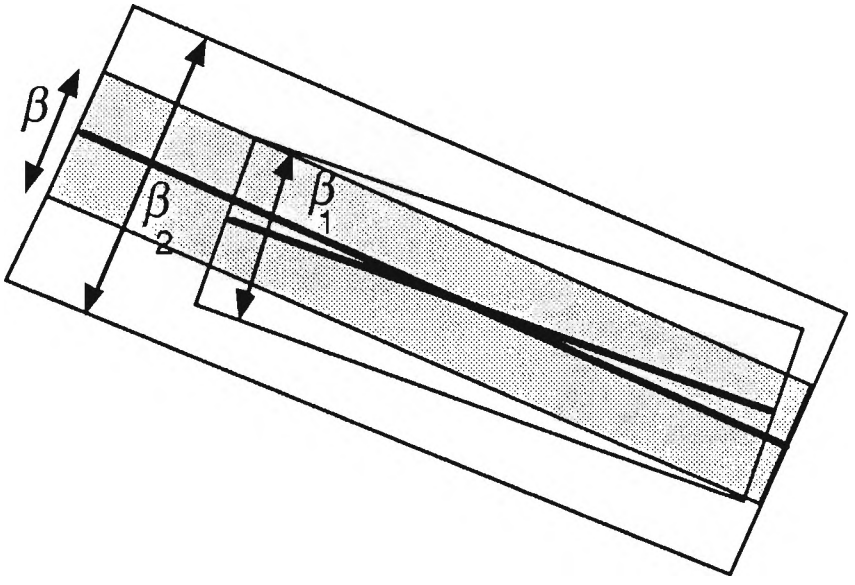


Fig. 8b. Updating the β region, after merging 2 collinear lines in the plane.

Incidence of a point and a line is decided as follows: (Fig 9 shows a point and a line with their ε and β regions). We define that a point and a line are *not* incident if their δ regions

have an empty intersection, otherwise we assume they are incident, and update the ε, β and δ regions of the point and the line.

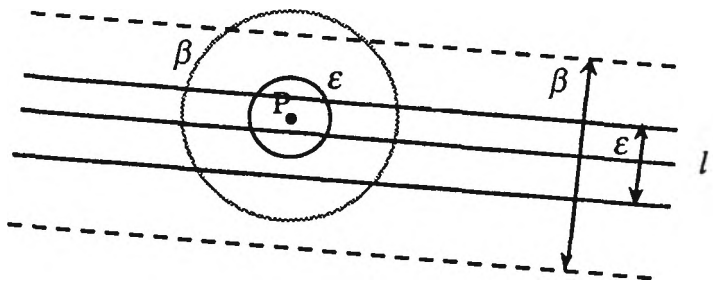


Fig. 9. Incidence of point and line with ε and β regions.

The radius of the ε circle of the point and the width of the ε rectangle are increased by the minimal amount, such that the lines of the rectangle become tangent to the circle. (see fig 10)

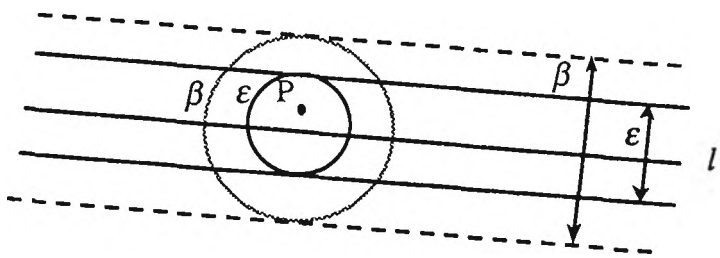


Fig. 10. The updated ε and β regions of the incident point and line.

The radius of the β circle of the point and the width of the β rectangle are decreased by the minimal amount such that the lines of the rectangle become tangent the β circle. The diameters of the ε and β circles of the point will be the same as the widths of the ε and β rectangles of the line. Updating the δ regions is handled the same way as the ε regions and is therefore not explicitly shown here. We define that a line and a point are incident, if the described updates can be made consistently (Note, that the ε and β circles are not necessarily concentric after the updates have been made). If the ε region is not completely contained in the β region after the update an ambiguity will be reported to the main algorithm.

Intersecting two lines should have the same outcome as computing the incidence of one point with two non collinear lines. Updating ε and β is therefore handled accordingly (an illustration is given in fig. 11). Note that the ε and β regions of the two lines will have equal width after intersecting. The δ regions of the lines will be updated the same way as the ε regions (not shown in fig. 11) such that they will have equal width afterwards.

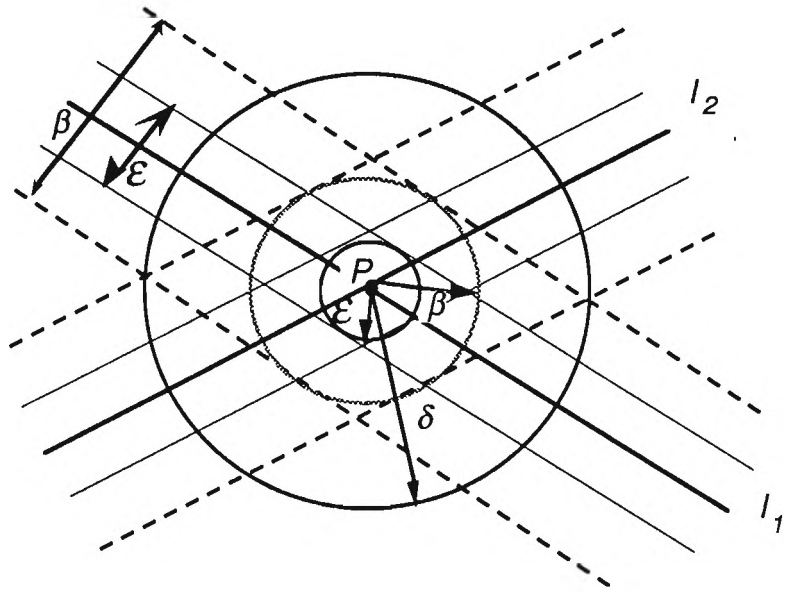


Fig. 11. Intersecting two lines with ε and β regions.

The δ region of the point will have at least the maximal extent of the intersection of the two β regions of the line. Thus, we take into consideration that intersecting two lines at a small angle is less accurate. In the extreme (when intersecting two collinear lines) the δ region would be infinitely large. Since collinear lines are merged and not intersected this case will not occur.

We now have to show that using these ε, β and δ regions consistently we never violate any law of planar Euclidean geometry.

Theorem 2 When computing collinearity of lines, incidence between points and lines and coincidence of points for a given, finite set of points and lines applying the described methods and data structures we either detect an ambiguity, or we can be sure that if the hypothesis of any first order theorem in Euclidean geometry expressed with these predicates, points and lines is satisfied then also the conclusion is satisfied.

Proof of Theorem 2 Wherever the algorithm detects an unambiguous degenerate case all the ε, β and δ regions of the involved elements will be updated, such that they are of equal with afterwards. Incident elements share the data for the ε, β and δ regions (see also section 6). Thus, we can ensure that a later change of one region gets propagated through all elements that are directly or indirectly related, and all the related ε and β regions will be of equal width. Therefore, there exist lines (one in the middle of the ε region of each line) and points (one in the middle of the ε region of each point) that satisfy the predicates exactly. If the points and lines occurring in an instance of the hypothesis of some theorem, and if they satisfy the conditions of that hypothesis, then the points and lines occurring in the conclusion can be constructed from these points and lines, and will also satisfy the conclusion of the theorem. Since these thought points and lines are totally within the ε regions of the points and lines, the ε regions of the points and lines in the conclusions will overlap. If on the other hand we found that the conclusion is not satisfied (because the δ regions do not overlap) then this means that either the hypothesis is not satisfied or somewhere the condition ($\varepsilon \subseteq \beta \subseteq \delta$) must have been violated during the algorithm and an ambiguity would have been reported. □

5.2 Linear Objects of Dimension Three and Higher

The uncertainty regions of objects should have simple shapes, complementing the dimensionality of the object in an n-dimensional space. They also should be symmetric in the complement space. For a point in a one-dimensional space these regions are intervals of length $2 \cdot \varepsilon$. For a point in 2D they are a circles with radius ε. In 3D the regions are spheres with a radius ε. The ε, β and δ regions of line segments in 3D have the form of cylinders (instead of the rectangles in 2D) (see fig. 12). The cylinders are bound on both sides, such that all points incident with a line are within the two ends of the cylinder. The uncertainty regions of planes are planar plates with thickness ε, β or δ, respectively. Again, we limit the valid region of the plane such that it contains all line segments and points incident with that plane. A simple way of approximating this is to define a cylindrical disc with height ε, β or δ, and a radius large enough to contain all the points and lines (see fig. 13).

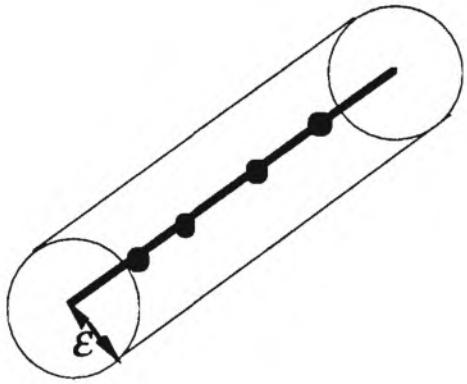


Fig. 12. The ε environment of a line in 3D.

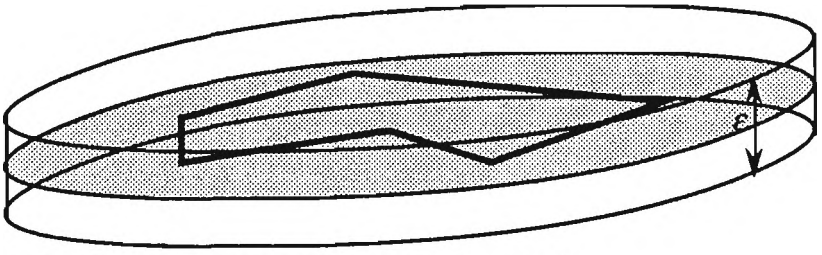


Fig 13. The ε environment of a plane in 3D.

For even higher dimensional linear objects (hyper planes and points) we can define the corresponding n-dimensional spheres and cylinders. Computing the relations and updating the ε, β and δ regions works similarly as in 2D. Basically, this means to carry out set operations (union and intersection) on these objects. To avoid possibly complicated geometric operations we approximate the result again by an n-dimensional sphere or cylinder, as we did it in 2D. A 3D application of this method is described in [BRU 90].

6 The Global Strategy of Ambiguity Testing and its Influence on the Complexity of Geometric Algorithms

One of the major advantages of this method is that it can be applied to most existing algorithms of computational geometry without the necessity of changing their logical structure. The algorithm can basically be run the same way as an algorithm that was originally designed for exact arithmetic. An upper bound for the round-off error must be estimated for every geometric operation. This additional data must be stored with the geometry information of each object in form of ε, β and δ regions. Whenever the algorithm computes a relation between two objects, the decisions and updates are made according the criteria described in the previous sections.

When the ε, β and δ regions of an object are updated we have to make sure that all its dependent objects get updated as well. This is achieved by letting objects that are related by a degeneracy condition share the same error data.

The *error data* of all objects are stored in records, separate from the objects. The records contain three spheres for ε, β and δ. For each of the spheres a center and a radius are stored. The centers of the spheres are defined relative to a point in the origin of the coordinate system.

A *point object* stores the vertex coordinates and a pointer to an error data record. Its error regions are defined by the spheres in the error data record offset by the coordinates of the vertex.

A *line object* stores a starting- and an end point and a pointer to an error data record. The uncertainty regions of a line are the cylinders obtained by sweeping the spheres of the error data record from start to end point.

Different objects related by a degeneracy condition have a pointer to the same error data, and they are linked together in a list. When comparing a new object with an object out of a set of related objects, in the case of a degeneracy, the error regions get updated, and the new object is inserted into the linked list of the related objects. When the new object already belongs to some other set of related objects we have to merge the two sets which means to redirect all the pointers of one set. To build a set of n related elements we have to do at most $n \log n$ pointer assignments.

If we can complete the algorithm without detecting ambiguous situations we know that the result is consistent (theorem 2). If on the other hand, the algorithm detects some ambiguity, this means that a consistent interpretation of the data with the specified tolerance is not possible. A way of dealing with detected inconsistencies is to increase the tolerance of the objects for which we detected the inconsistency. We would have to rerun the algorithm with the new ε, δ and β regions. Starting all over again is necessary because the new tolerances might produce new inconsistencies in some other parts of the algorithm. We may have to iterate this procedure until we encounter no more ambiguous situations. When we start with ε regions big enough to embrace the largest accumulated error for the respective objects we have a high chance of finding a consistent interpretation the first time. An

ambiguity will only be detected when a generic case comes to lie very close to the degenerate case and is erroneously taken for a "real" degenerate cases. For this reason, comparing relatively exact data with relatively inexact data leads to undecidable comparisons when the ε regions overlap. In this case it is recommended to assign the same tolerance to all objects in the model. Sometimes, the data is ambiguous in the sense that no clear distinction between degenerate cases and generic cases can be made. In the example of set operations on polyhedra we can provoke this situation by intersecting two identical cubes that are slightly rotated against each other by a small angle with just about the amount of the tolerance for angles (such an example is described in [HHK 88]). Comparing the planes results in ambiguities. Ignoring them would likely lead to dangling edges or faces in the resulting object. After making the ε for the uncertainty of planes somewhat larger the almost coplanar planes will be considered coplanar, and the two cubes will be considered identical. The intersection of the two cubes would be identical to one of the two cubes intersected which is consistent with the larger tolerance introduced. In this example the algorithm succeeds in the second run. In search for the worst case, we consider the example where n points are lined up in a row. Each point has a distance slightly smaller than ε from its nearest neighbor. When comparing these points pairwise we would find that each point coincides with its nearest neighbor, but points that are farther apart are considered distinct. Since every point has one neighbor that is considered equal, and they are all on a line, they are all identical because of the transitivity law. When updating the ε, β and δ regions we will detect an inconsistency. We can rerun the algorithm with a new ε that is, for instance, twice as large for each point. Still there would be points that are farther away, such that they are considered distinct. We would have to rerun the algorithm again and again with ever increasing ε values. The algorithm will eventually stop when ε is at least as large as the distance of the two points on the opposite ends of the line. In other words, the uncertainty is as large as the whole model, and all points are identical. This result reminds us of the fact that an approximation of real objects by floating point numbers has some inherent limitations.

What is the impact of this method on the complexity of some algorithm in the worst case? We assume that the time complexity of the original algorithm is $O(f(n))$, where n is the total number of primitives (points, edges, planes, etc.). In one run of the algorithm, applying the ambiguity test increases the time by a constant factor and adds time $n \log n$ for the pointer operations in the worst case. For most algorithms $n \log n$ is a lower bound of complexity. For these algorithms the complexity is not changed by the ambiguity test. When an ambiguity is detected the algorithm will need to be run again with larger ε-values.

In the worst case we find one additional degenerate case, and therefore add one new element to the closure of relation of a certain type. In the example of the n points in a row we might have to run the algorithm up to n times until we find a large enough tolerance that leads to a consistent result. In most practical cases, however, the algorithm will have to be run once only, sometimes twice, to produce a consistent result.

7 Conclusion and Future Plans

The greatest advantage of the approach presented here is that it delivers consistent results although we employ limited accuracy floating point variables and operations. This is usually preferable to the much slower rational arithmetic, or the complicated and time consuming symbolic reasoning about the logical dependencies in the data used by some approaches. The topological properties of the geometric objects are not changed as this is the case with the perturbation method which avoids degeneracies, or with the line cracking method which introduces new points and lines to avoid inconsistencies (e.g. [MIL 88]). The approach described here allows the algorithm to smooth out small differences where intended (e.g., approximately parallel planes become parallel, approximately coincident points become identical). Therefore, degenerate cases can really be treated as degenerate cases where this is desired.

The decision mechanism can be easily added to already existing computational geometry algorithms without changing their underlying structure. It is necessary to estimate an upper bound for round-off errors for each geometric operation, and to carry out some relatively simple book keeping on the accuracy intervals after each comparison. This does not significantly slow down the algorithm in most cases. Sometimes, however, the algorithm must be rerun several times to obtain a consistent result. In practice, most problems are well behaved in the sense that a clear distinction between degenerate and generic cases can be made. In these cases the algorithm is approximately as fast as an algorithm that does not check for inconsistencies. Another approach that is also based on floating point arithmetic is called "ε-geometry" [GSS 89]. One of the major differences to this paper is that in ε-geometry the uncertainty is associated with predicates (collinearity, incidence, etc.), rather than with objects. Associating ε with the objects has the advantage that no reasoning about the dependencies of the predicates is required. The objects themselves "remember" the decisions made about them, and decide whether or not they are consistent. Therefore, it is not even necessary to know which theorem needs to be satisfied to make a specific algorithm consistent. It is possible, however, that we pay the price for such generality in form of an ε that sometimes grows faster than really necessary.

The advantages make this approach well suited for complicated interactive geometric modelling applications. A solid modelling algorithm, based on this approach is described in [BRU 89]. Set operations are among the most complicated operations in geometric modelling. It is very difficult to prove that a heuristic approach to inference dependencies works correct in any case (see [HHK 88]).

In the future, we would like to find an adaptive method that does not require to rerun the whole algorithm if some ambiguity occurs, but rather tries to fix the problem locally. This does not seem to be so easy, and probably requires changing the structure of the algorithms in most applications. This in turn, would probably have a negative influence on the efficiency when degenerate cases occur. Also, we would like to apply this methods for robustly intersecting spline curves and sculptured surfaces. This seems possible, however, it is crucial for the success of this method to employ a good criterion for estimating upper bounds for errors. This seems difficult when an intersection curve is represented by a piecewise linear approximation.

Acknowledgements

I would like to thank Elaine Cohen, Michael Cohen, Shiaofen Fang, Surya Mantha, Rok Sosič and Frank Stenger for their helpful remarks on an earlier version of this paper. This work was supported in part by NSF grant # DDM-8910229.

References

[ALH 83] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, New York, 1983

[BCK 88] B. Buchberger, G. Collins, and B. Kutzler, *Algebraic Methods for Geometric Reasoning*. Ann. Reviews in Computer Science, Ann. Reviews Inc., Palo Alto, Calif., Vol. 3, 1988, pp. 85 - 119

[BRU 90] B. Brüderlin. *Robust Regularized Set Operations on Polyhedra*. Tech Report UUCS-9004, Univ. of Utah, April 1990

[BRU 88] B. Brüderlin. *Automatizing Geometric Proofs and Constructions*. In Computational Geometry and its Applications, H. Noltemeier (Ed), Springer Lecture Notes in Computer Science No 333, pp. 232 - 252, Springer Verlag, 1988

[CHO 88] S.C. Chou. *Mechanical Geometry Theorem Proving*. D. Reidel Publishing Company, Dordrecht, Netherlands, 1988

[GSS 89] L. Guibas, David Salesin, Jorge Stolfi. Epsilon Geometry: Building Robust Algorithms from Imprecise Computations. 5th Annual ACM Symposium on Computational Geometry, ACM, 1989

[EDM 88] H. Edelsbrunner and E. Mücke. *Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms"*Proc. Fourth ACM symp. on Computer Geometry, June 1988, pp. 118 - 133

[HHK 88] Christoph M. Hoffmann, John E. Hopcroft, Michael S. Karasick. *Robust Set Operations on Polyhedral Solids*. Technical Report 723, Computer Science Dept., Purdue University, W. Laffayette, Ind., 1988

[HOF 89] Christoph M. Hoffmann. *The Problems of Accuracy and Robustness in Geometric Computation.* IEEE Computer, Volume 22, No. 3, March '89

[MIL 88] V. Milenkovic. *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic*. Tech. Report CS-88-168, Carnegie Mellon Univ., Pittsburgh, 1988

[OTU 87] Th. Ottmann, G. Thiemt, Ch. Ullrich. *Numerical Stability of Geometric Algorithms*. Proc. 3rd ACM Symposium on Computational Geometry, 1987

[SES 88] M. Segal and C. Séquin. *Consistent Calculations for Solids Modeling. Artificial Intelligence*. D. Kapur and J. Mundy, eds. Special issue on geometric reasoning, Elsevier, North Holland, Vol. 37, 1988

[YAP 88] C. Yap. *A Geometric Consistency Theorem for a Symbolic Perturbation Theorem*. Proc. Fourth ACM Symposium on Computer Geometry, June 1988